

Using the Twitter REST API

Explore the Twitter REST API for automated Web 2.0

Skill Level: Intermediate

[Brian M. Carey \(careyb@triangleinformationsolutions.com\)](mailto:careyb@triangleinformationsolutions.com)

Information Systems Consultant
Triangle Information Solutions

09 Jun 2009

Twitter is undoubtedly one of the most recent and successful examples of social networking to appear on the World Wide Web. Twitter provides an API so Web developers can enable their users to access the various features that the Twitter site provides. In this article, learn the basics of using the Twitter REST API.

Twitter is simply a Web-based way to tell certain people know what you are currently doing in 140 characters or less.

That's the short definition.

The long definition is a bit more involved, but it does merit consideration. Twitter is one of the most successful entries in what the industry now refers to as *social media*, *online social networking*, or *Web 2.0*. Using Twitter, you gather a number of followers. Then, from time to time, to tell them what you are doing, you type a little story (known as a *tweet* in the industry) in the Twitter GUI and click a button. That tweet is then transmitted to all of your followers, and they can read, understand, reply, or not care accordingly.

Frequently used acronyms

- API: Application programming interface
- GUI: Graphical user interface
- HTTP: Hypertext Transfer Protocol
- IP: Internet Protocol

- REST: Representational State Transfer
- RSS: Really Simple Syndication
- URL: Uniform Resource Location
- XML: Extensible Markup Language

Shakespeare tells us that "brevity is the soul of wit." This philosophy is enforced by the Twitter authorities, as tweets are limited to a maximum of 140 characters. Actually, that limitation has nothing to do with Shakespeare: It has to do with limitations on mobile devices at the time Twitter was developed. But it is a welcome enforcement, as it prevents unnecessary spam and verbal clutter within a single tweet.

Although the length of tweets is strictly enforced, the actual content of those tweets is not so strictly enforced. The original intent of Twitter was to tell your followers what you are doing right now. Needless to say, that is not always the subject of the millions of tweets issued daily. People will post opinions, headlines, links to their blogs, links to someone else's blog, and so on. So, new users of Twitter should be prepared to receive tweets that have nothing to do with the tweeter's current task.

Twitter also comes with an additional benefit associated with most (if not all) of Web 2.0: It is free. That's right, it doesn't cost you anything to join. It doesn't cost you anything to follow someone else. It doesn't cost you anything to have any number of followers. It doesn't cost you anything to tweet. It's just there for your consumption.

By now, you have a broad overview of Twitter and what it does. If you have not yet visited the [Twitter site](#), now is a good time before you go on with the rest of the article. It will be much easier to understand the REST API that way.

The Twitter REST API

Having covered the basics, you're ready to move on to the stuff that Web application developers enjoy. Twitter is not only a useful tool within the social media space, it also offers developers a comprehensive array of services to enable automation of Twitter functionality. One of those services (and perhaps the most popular) is the REST API.

REST is an acronym for *Representational State Transfer*. The full explanation of everything entailed in a proper REST definition is outside of the scope of this article; however, it is available elsewhere on IBM® developerWorks® (see [Resources](#)). For the subject covered here, it is sufficient to state that REST enables developers to access information and resources using a simple HTTP invocation.

As an example, imagine that FishinHole.com operates a Web site that markets

fishing tackle to its customers. Users who access the site can see a variety of lures, reels, rods, and so forth. They do this the old-fashioned way: by clicking links. In this way, FishinHole.com makes its services available to human beings.

But FishinHole.com also makes its services available to other Web applications by exposing its catalog of fishing tackle with REST. So, instead of clicking around, the Web application obtains information about lures, reels, rods, and so forth with a simple HTTP invocation. For example, `http://www.fishinhole.com/catalog/rest/lures?format=xml` returns a list of all lures offered by the company in XML format. As another example, `http://www.fishinhole.com/catalog/rest/item?id=343221` returns information about item #343221 in the default format.

Think of REST this way: you can obtain domain-specific data simply by pointing a URL to a specific location. For our purposes here, that's really all it is. You can also think of it as a simplified Web service, but if you say that too loudly around the wrong people, you might find yourself in the middle of a debate.

Note: I should point out that FishinHole.com doesn't actually exist. So, if you paste any of those URLs into your browser, you might wonder why you got an error. I provide these examples simply to show the format of a typical REST invocation.

Would you like to see an example of a fully functioning REST API? One where you can actually paste URLs into a browser and get something significant returned? Then please read on.

Getting started: a simple example

You just finished reading Joel Comm's great book, *Twitter Power*, and you decide that today you will begin to achieve financial independence with an aggressive online marketing program using Twitter.

But you're also a great software developer. And that means that you prefer to let software do much of the work for you rather than actually do a lot of the Twittering yourself. Not only do you register a new Twitter account, but you also start to learn about the API so you can automate certain aspects of Twitter functionality.

The first thing you want to do is use the API to retrieve Joel Comm's timeline (see [Listing 1](#)). This makes sense, because he wrote the book that was such an inspiration to you.

Listing 1. Retrieving Joel Comm's timeline

```
http://twitter.com/statuses/user_timeline.xml?id=joelcomm
```

That's it. It's just that simple. Open another browser, paste that URL into the address bar, and see what you get.

Obviously, a closer examination of that REST invocation is warranted. First, the `http://twitter.com` prefix should be self-explanatory. The `twitter.com` portion is the domain name, which indicates that you will access a resource at an IP address to which that name is mapped. The `http` that precedes it indicates that you will use the Hypertext Transfer Protocol. This is frequently the case with REST.

Next comes `/statuses`. This is how Twitter specifies the REST function within a particular category. Think of it like a directory within a file system. In this case, the REST function that is invoked is categorized under `statuses`. In Twitter terminology, a user *status* is basically a tweet, as it tells you what the user is doing right now.

Next comes `user_timeline`. This is the actual function name to be invoked. This is intuitively named `user_timeline` because, in fact, you are retrieving a user *timeline*, or a series of tweets that the user has recently entered.

Don't miss the `.xml` extension that follows the function name: This is important. It is the format in which the timeline will be retrieved. Here, you use XML. Other valid formats are Java™ Simple Object Notation (JSON), Atom, and RSS.

Using standard `GET` notation, the parameters follow the function and are delimited by a question mark (`?`). In this case, there is only one parameter—`id`—and it specifies the Twitter name of the user whose timeline you want to view. Here, `joelcomm` is specified, because he is the one whose timeline you want to see.

Evaluating the output

You realize after viewing the output from above that you prefer to receive your results in Atom format. Fortunately, this is not a problem at all and only requires a minor change ([Listing 2](#)) to the code in [Listing 1](#).

Listing 2. Retrieving Joel Comm's timeline in Atom

```
http://twitter.com/statuses/user_timeline.atom?id=joelcomm
```

The REST invocation above yields something similar to the results in [Listing 3](#). If you paste that code into your URL, your browser might ask you to download the resulting output, because your browser is not configured to display files ending with the `.atom` extension.

Obviously, Joel's timeline will be different when this article goes to press (and at the time you read this) versus his timeline while I wrote this article. So the exact results

will certainly vary.

Listing 3. Joel Comm's timeline in Atom format (abbreviated)

```
<?xml version="1.0" encoding="UTF-8"?>
<feed xml:lang="en-US" xmlns="http://www.w3.org/2005/Atom">
  <title>Twitter / joelcomm</title>
  <id>tag:twitter.com,2007:Status</id>
  <link type="text/html" rel="alternate" href="http://twitter.com/joelcomm"/>
  <updated>2009-03-22T10:21:31+00:00</updated>
  <subtitle>Twitter updates from Joel Comm / joelcomm.</subtitle>
  <entry>
    <title>joelcomm: thinking...</title>
    <content type="html">joelcomm: thinking...</content>
    <id>tag:twitter.com,2007:http://twitter.com/joelcomm/statuses/1369295498</id>
    <published>2009-03-22T05:15:01+00:00</published>
    <updated>2009-03-22T05:15:01+00:00</updated>
    <link type="text/html" rel="alternate"
      href="http://twitter.com/joelcomm/statuses/1369295498"/>
    <link type="image/jpeg" rel="image"
      href="http://s3.amazonaws.com/joell_normal.jpg"/>
    <author>
      <name>Joel Comm</name>
      <uri>http://www.JoelComm.com</uri>
    </author>
  </entry>
</feed>
```

If you are familiar with XML, you'll find that most of [Listing 3](#) is intuitive. If you're familiar with Atom, you'll find it even more familiar. If you're familiar with Atom *and* Twitter, you can probably skip this section.

Here's the breakdown on the code in [Listing 3](#):

- Note that the root element is `feed`. This is standard according to the Atom specification. The namespace that Twitter uses is `http://www.w3.org/2005/Atom`, as specified as an attribute in the root element.
- The `title` element identifies the user whose timeline you are viewing. It also provides a bit of advertising for the Twitter Web site.
- The `link` element is also important: It specifies the URL you use if you want to view Joel Comm's timeline the old-fashioned way (by manually viewing it in your browser).
- The `entry` stanza represents a tweet. Although for the sake of brevity I only list one, in reality, you will see 20 of these in your output.
- Notice that `title` and `content` are the same in actual content. This is because tweets have no titles, so it makes sense that the title is the actual tweet itself. Recall that Atom is designed for article-type documents, which usually have a headline, then a main body. Because that is not the case with tweets, the two elements contain identical content.

- In Atom format, the content is preceded by the Twitter name, then a colon (:). Here, `joelcomm:` precedes the actual tweet.
- The actual tweet here is the oh-so-significant statement `thinking...`. That's Joel's latest tweet as I write this article. A cynical individual might infer that this indicates that there are certain times when Joel is *not* thinking or that Joel was lacking material for his latest tweet and felt the urge to simply type something. However, I leave such suppositions to others.
- The `id` element is required by Atom and is a globally unique identifier (GUID) for this particular tweet. All tweets across the universe of Twitter will have unique IDs so that they can be referenced individually.
- The `published` and `updated` date and times are also identical. This makes sense, because Joel simply entered his tweet and never updated it.
- The first `link` element provides a link to this single tweet. Go ahead and paste `http://twitter.com/joelcomm/statuses/1369295498` into a browser window, and you'll see that Joel was "thinking..." at that time.
- The second `link` element simply provides a link to Joel's picture.
- The `author` stanza provides information about the Twitter user. Here, you see Joel's full name and Web site URL.

After contemplating this much of the API, you realize that this is fabulous information and that you can easily write code to parse the Atom output. You, of course, can also parse time lines from other power users, not just Joel Comm. The parsed information can be gleaned for data relevant to your online marketing campaign. The only limitation is your imagination: the possibilities are endless.

Other parameters

The `user_timeline` has several parameters in addition to `id`. You can also specify `screen_name` instead of `id` in the case above. If you happen to know the user's numerical Twitter ID, you can specify that in the `user_id` parameter.

You can use the `since_id` parameter to specify tweets with an ID higher than the number specified in this parameter (see [Listing 4](#)). Recall from above that Joel's famous "thinking..." tweet had an ID of 1369295498. So, the following URL returns tweets later than that one.

Listing 4. Retrieving Joel Comm's timeline since "thinking..."

```
http://twitter.com/statuses/user_timeline.xml?id=joelcomm&since_id=1369295498
```

The parameter `max_id` is basically the reverse of `since_id`. It returns tweets with IDs *less than* the one specified by the parameter value.

The parameter `since` allows you to apply an actual date to your timeline filter, as opposed to an ID. The `page` parameter allows you to paginate your results. A default `user_timeline` invocation returns the last 20 tweets. If you were to number these tweets 1-20, then the code in [Listing 5](#) returns tweets 41-60.

Listing 5. Retrieving Joel Comm's third set of 20 tweets

```
http://twitter.com/statuses/user_timeline.xml?id=joelcomm&page=3
```

Other functions

Thus far, you've extensively examined the `user_timeline` function. However, the Twitter API provides other functions that are accessible through REST.

The `public_timeline` function ([Listing 6](#)) allows you to see the latest tweets across the entire Twitterverse—at least for those users who make their tweets publicly available.

Listing 6. The latest tweets

```
http://twitter.com/statuses/public_timeline.xml
```

The `friends_timeline` function ([Listing 7](#)) allows you to see the tweets of people that you follow. This is the same as if you log in to Twitter and go straight to your Twitter home page.

Listing 7. The latest tweets of people you are following

```
http://twitter.com/statuses/friends_timeline.xml
```

If you copy and paste the URL in [Listing 7](#) into your browser, you might notice that you are prompted to supply your Twitter user name and a password. Your home page in Twitter is a secure environment, as it contains links to your direct messages. So, this is a security measure on the part of Twitter. (I discuss security in more detail later in this article.)

The `update` function allows you to actually tweet using the REST API. In this case,

the function invocation must be accomplished using a `POST` request (as opposed to a `GET` request). The parameter `status` submitted with the `POST` request contains the text of the actual tweet.

The `replies` function returns the 20 most recent `@replies` to the authenticated user. Basically, `@replies` are tweets directed specifically at a particular user. For example, if you tweet `@joelcomm` are you done thinking yet?, that message appears as one of a series of messages directed in particular to Joel Comm. He can see these messages by clicking a link on his Twitter home page. However, `@replies` are also visible to all users following the user who issued the reply.

It is beyond the scope of this article to explain all the REST API functions in complete detail. However, they are clearly documented in the API documentation. For a link to that documentation, see [Resources](#).

Limitations on API use

Using the Twitter REST API is not a *carte blanche* to do whatever you want. Twitter has placed certain restrictions on the use of its API to prohibit bandwidth hogs from jeopardizing the usefulness of the feature set.

For starters, you are only allowed a maximum of 100 requests per hour. Although this limit applies only to `GET` (as opposed to `POST`) requests, it is still a good rule of thumb to follow. If you exceed the limit, your resultant document from the REST invocation tells you so. If, for whatever reason, you *must* invoke the Twitter REST API more than 100 times per hour, you can request *whitelisting* from Twitter.

Another limitation is that a maximum of 3200 statuses can be returned when using either the `page` or `count` parameters.

In addition, Twitter requests but does not demand other limitations. For example, Twitter advises that you use the `page` attribute over the `count` attribute. The company also asks that you cache results locally as opposed to requesting the same status repeatedly.

Authentication

As I mentioned earlier, certain functions require authentication. If you want to use the Twitter REST API and take advantage of those functions, you must include your credentials in the request. Otherwise, you will get a status code 401 for your reply.

As of this writing, Twitter only supports HTTP basic authentication, which means that the request header must contain your user name and password in an encrypted format. You will then have full access to the Twitter API functions as though you

logged in to Twitter from your browser. For more information about basic authentication, see [Resources](#).

Currently, Twitter is working on a way to enable OAuth authentication for secured requests. As of this writing, that is still in development.

Conclusion

Twitter is a fabulous entry in the Web 2.0 genre. Using Twitter, you can microblog your way to building an entire online network of individuals who share common interests with you.

Using the Twitter REST API, you can automate just about everything you can do with Twitter manually. You can programmatically access a specific user's timeline. You can reply to that user, either directly or indirectly. You can search a user's tweets for information specific to your own interests. You can filter tweets based on certain criteria and display those tweets on your own blog.

The possibilities are endless.

Resources

Learn

- [Build a RESTful Web service](#) (Andrew Glover, developerWorks, July 2008): Read an excellent explanation of Representational State Transfer in an introduction to REST and the Restlet framework.
- [RESTful Web services: The basics](#) (Alex Rodriguez, developerWorks, November 2008): Discover another excellent overview of REST and its basic principles.
- [HTTP Basic authentication](#): Start with the Wikipedia Basic authentication entry, a great place to learn about Basic authentication.
- [Twitter REST API documentation](#): Get a comprehensive overview of the entire API, complete with examples.
- [IBM XML certification](#): Find out how you can become an IBM-Certified Developer in XML and related technologies.
- [XML technical library](#): See the developerWorks XML Zone for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks.
- [developerWorks technical events and webcasts](#): Stay current with technology in these sessions.
- [developerWorks podcasts](#): Listen to interesting interviews and discussions for software developers.

Get products and technologies

- The [Twitter site](#): Explore the Twitter service. Try it and be connected with friends, family, and co-workers as you exchange short messages about what you do.
- [IBM product evaluation versions](#): Download or [explore the online trials in the IBM SOA Sandbox](#) and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- [XML zone discussion forums](#): Participate in any of several XML-related discussions.
- [developerWorks blogs](#): Check out these blogs and get involved in the [developerWorks community](#).

About the author

Brian M. Carey

Brian Carey is an information systems consultant who specializes in the architecture, design, and implementation of Java enterprise applications. You can follow Brian on Twitter at <http://twitter.com/brianmcarey>, and his tweets are publicly available.

Trademarks

IBM, the IBM logo, ibm.com, DB2, developerWorks, Lotus, Rational, Tivoli, and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. See the current list of [IBM trademarks](#).

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.